

Webscraping I

Peter Ganong and Maggie Shi

November 4, 2024

Table of contents I

Introduction to HTML and Web Scraping

Using BeautifulSoup to Parse HTML

Example: Applying BeautifulSoup to a Website

Introduction to HTML and Web Scraping

Roadmap:

- ▶ Intro to webscraping
- ▶ Intro to HTML
 - ▶ Simple example
 - ▶ Harris website example
- ▶ Do-pair-share

Webscraping

- ▶ **Webscraping** uses code to systematically extract content and data from websites
- ▶ Though websites vary a lot in how they're structured and where data is located, most are constructed using a common language: HTML
- ▶ Each website can be converted into its underlying HTML code and then parsed with Python

Webscraping

The steps of building a webscraper are:

0. *Manual*: inspect website's HTML to see how the info we want to extract is structured
1. *Code*: download and save HTML code associated with a URL
2. *Code*: parse through HTML code to extract information based on what we learned in Step 0 + refine
3. *Code*: organize and save extracted information

Webscraping

The steps of building a webscraper are:

0. *Manual*: inspect website's HTML to see how the info we want to extract is structured
1. *Code*: download and save HTML code associated with a URL
2. *Code*: parse through HTML code to extract information based on what we learned in Step 0 + refine
3. *Code*: organize and save extracted information

So before we learn how to code a web scraper, we need to **understand how to read HTML code**

Intro to HTML

- ▶ **HTML:** Hypertext Markup Language
- ▶ Tells your web browser how to display the content of a web page

Intro to HTML

- ▶ **HTML:** Hypertext Markup Language
- ▶ Tells your web browser how to display the content of a web page
- ▶ Structure:

```
<name_of_tag attribute1 = 'value'> content <\name_of_tag>
```

1. **Tags:** keyword that defines what element is, such as text, paragraph, heading, link, etc.
 2. **Attributes:** additional information about element
 3. **Content:** text, images, or other media associated with that element
- ▶ HTML is structured hierarchically, so tags can be nested within tags

simple.txt example

File: simple.txt

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to my webpage</title>
</head>

<body>
<h1 id='first'>Today we start <em>coding!</em></h1>
<h2 class='myclass'>Hello world</h2>
<p style="color: green;">Hello world, but in green.</p>

<p><a href='https://en.wikipedia.org/wiki/Dog'>click for
dogs</a></p>
<p id = 'second'> Goodbye!</p>

</body>
</html>
```

simple.txt example

File: simple.txt

opening tag

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to my webpage</title>
</head>

<body>
<h1 id='first'>Today we start <em>coding!</em></h1>
<h2 class='myclass'>Hello world</h2>
<p style="color: green;">Hello world, but in green.</p>

<p><a href='https://en.wikipedia.org/wiki/Dog'>click for
dogs</a></p>
<p id = 'second'> Goodbye!</p>
```

closing tag

```
</body>
</html>
```

simple.txt example

File: simple.txt



simple.txt example

File: simple.txt

Head:
metadata

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to my webpage</title>
</head>
```

Body:
browser content

```
<body>
<h1 id='first'>Today we start <em>coding!</em></h1>
<h2 class='myclass'>Hello world</h2>
<p style="color: green;">Hello world, but in green.</p>

<p><a href='https://en.wikipedia.org/wiki/Dog'>click for
dogs</a></p>
<p id = 'second'> Goodbye!</p>

</body>
</html>
```

simple.txt example

File: simple.txt

Head:
metadata

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to my webpage</title>
</head>
```

Relative to **<head>**, this is the content

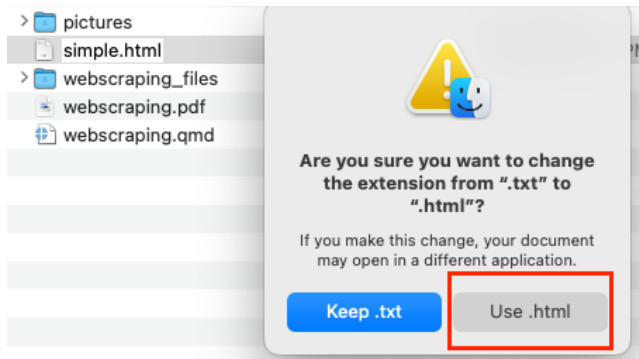
Body:
browser content

```
<body>
<h1 id='first'>Today we start <em>coding!</em></h1>
<h2 class='myclass'>Hello world</h2>
<p style="color: green;">Hello world, but in green.</p>
<p><a href='https://en.wikipedia.org/wiki/Dog'>click for
dogs</a></p>
<p id = 'second'> Goodbye!</p>
</body>
</html>
```

Relative to **<body>**,
this is the content

simple.txt example

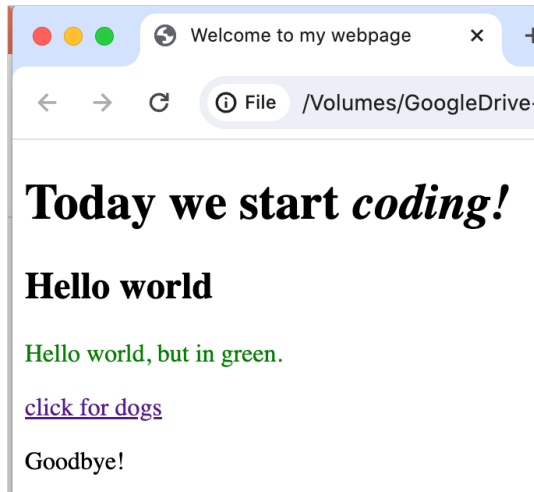
To see the HTML in action, rename the file extension from .txt to .html



Click "Use .html" when prompted

simple.txt example

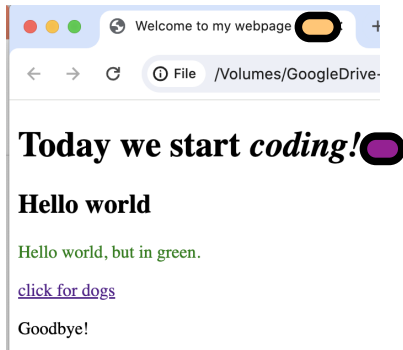
This should open as a web page in your default web browser



simple.txt example

File: simple.txt

`<name_of_tag attribute='value'>Content<\name_of_tag>`



```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to my webpage</title>
</head>

<body>
<h1 id='first'>Today we start <em>coding!</em></h1>
<h2 class='myclass'>Hello world</h2>
<p style="color: green;">Hello world, but in green.</p>

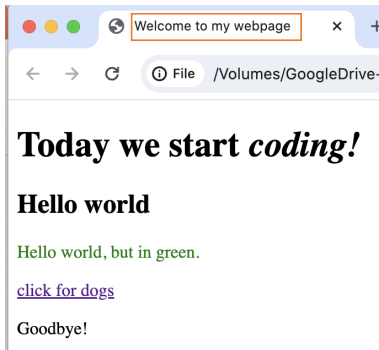
<p><a href='https://en.wikipedia.org/wiki/Dog'>click for
dogs</a></p>
<p id = 'second'> Goodbye!</p>

</body>
</html>
```

simple.txt example

File: simple.txt

`<name_of_tag attribute='value'>Content<\name_of_tag>`



```
<!DOCTYPE html>
<html>
  <head>
    <title>Welcome to my webpage</title>
  </head>

  <body>
    <h1 id='first'>Today we start <em>coding!</em></h1>
    <h2 class='myclass'>Hello world</h2>
    <p style="color: green;">Hello world, but in green.</p>

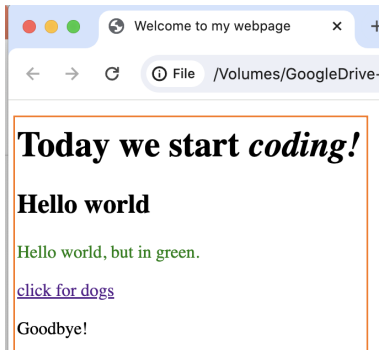
    <p><a href='https://en.wikipedia.org/wiki/Dog'>click for
    dogs</a></p>
    <p id = 'second'> Goodbye!</p>

  </body>
</html>
```

simple.txt example

File: simple.txt

`<name_of_tag attribute='value'>Content<\name_of_tag>`



```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to my webpage</title>
</head>

<body>
<h1 id='first'>Today we start <em>coding!</em></h1>
<h2 class='myclass'>Hello world</h2>
<p style="color: green;">Hello world, but in green.</p>

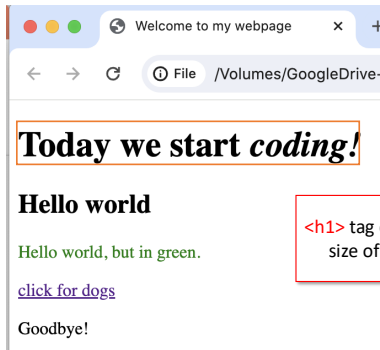
<p><a href='https://en.wikipedia.org/wiki/Dog'>click for
dogs</a></p>
<p id = 'second'> Goodbye!</p>

</body>
</html>
```

simple.txt example

File: simple.txt

`<name_of_tag attribute='value'>Content<\name_of_tag>`



```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to my webpage</title>
</head>

<body>
<h1 id='first'>Today we start <em>coding!</em></h1>
<h2 class='myclass'>Hello world</h2>
<p style="color: green;">Hello world, but in green.</p>
<a href='https://en.wikipedia.org/wiki/Dog'>click for
dogs</a></p>
<p id = 'second'> Goodbye!</p>

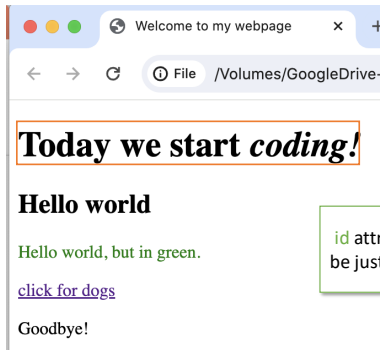
</body>
</html>
```

`<h1>` tag determines
size of the text

simple.txt example

File: simple.txt

`<name_of_tag attribute='value'>Content<\name_of_tag>`



```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to my webpage</title>
</head>

<body>
<h1 id='first'>Today we start <em>coding!</em></h1>
<h2 class='myclass'>Hello world</h2>
<p style="color: green;">Hello world, but in green.</p>
<a href='https://en.wikipedia.org/wiki/Dog'>click for
a</a></p>
<p id = 'second'> Goodbye!</p>

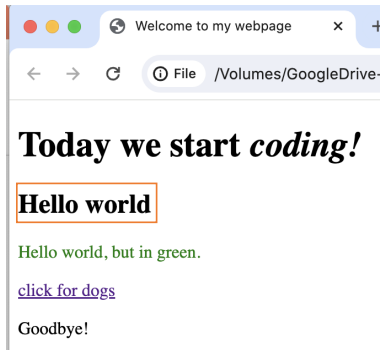
</body>
</html>
```

id attribute: should
be just one per page

simple.txt example

File: simple.txt

`<name_of_tag attribute='value'>Content<\name_of_tag>`



```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to my webpage</title>
</head>

<body>
<h1 id='first'>Today we start <em>coding!</em></h1>
<h2 class='myclass'>Hello world</h2>
<p style="color: green;">Hello world, but in green.</p>

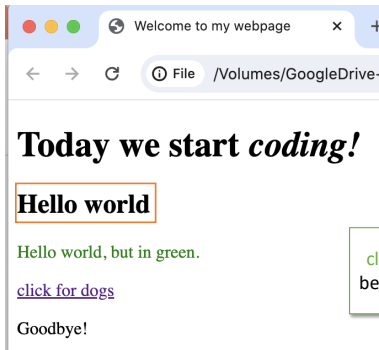
<p><a href='https://en.wikipedia.org/wiki/Dog'>click for
dogs</a></p>
<p id = 'second'> Goodbye!</p>

</body>
</html>
```

simple.txt example

File: simple.txt

`<name_of_tag attribute='value'>Content<\name_of_tag>`



```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to my webpage</title>
</head>

<body>
<h1 id='first'>Today we start <em>coding!</em></h1>
<h2 class='myclass'>Hello world</h2>
<p style="color: green;">Hello world, but in green.</p>
<p class="first" id="second" f='https://en.wikipedia.org/wiki/Dog'>click for
second'> Goodbye!</p>

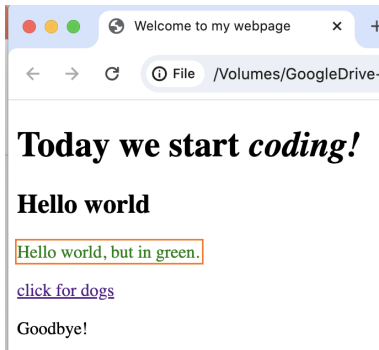
</body>
</html>
```

class attribute: can
be multiple per page

simple.txt example

File: simple.txt

`<name_of_tag attribute='value'>Content<\name_of_tag>`



```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to my webpage</title>
</head>

<body>
<h1 id='first'>Today we start <em>coding!</em></h1>
<h2 class='myclass'>Hello world</h2>
<p style="color: green;">Hello world, but in green.</p>

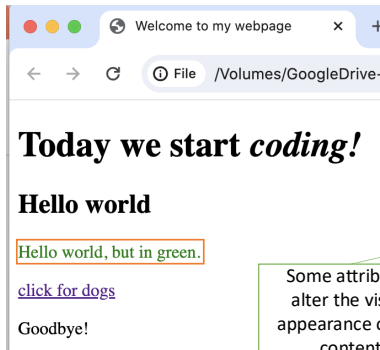
<p><a href='https://en.wikipedia.org/wiki/Dog'>click for
dogs</a></p>
<p id = 'second'> Goodbye!</p>

</body>
</html>
```


simple.txt example

File: simple.txt

`<name_of_tag attribute='value'>Content<\name_of_tag>`



```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to my webpage</title>
</head>

<body>
<h1 id='first'>Today we start <em>coding!</em></h1>
<h2 class='myclass'>Hello world</h2>
<p style="color: green;">Hello world, but in green.</p>
<p><a href='https://en.wikipedia.org/wiki/Dog'>click for
dogs</a></p>
<p id = 'second'> Goodbye!</p>

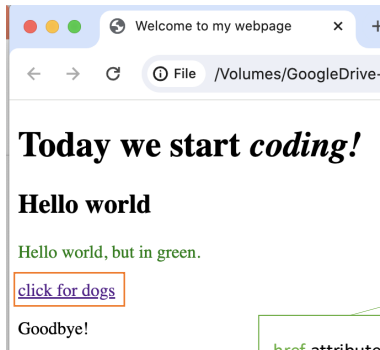
</body>
</html>
```

Some attributes
alter the visual
appearance of the
content

simple.txt example

File: simple.txt

`<name_of_tag attribute='value'>Content<\name_of_tag>`



```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to my webpage</title>
</head>

<body>
<h1 id='first'>Today we start <em>coding!</em></h1>
<h2 class='myclass'>Hello world</h2>
<p style="color: green;">Hello world, but in green.</p>

<p><a href='https://en.wikipedia.org/wiki/Dog'>click for
dogs</a></p>
<p id = 'second'> Goodbye!</p>

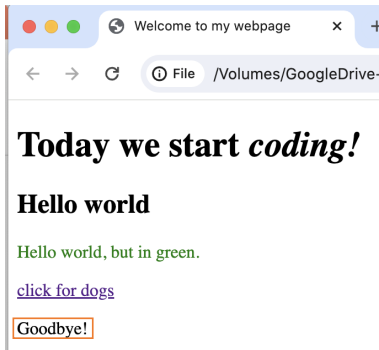
</body>
</html>
```

href attribute turns
it into a hyperlink

simple.txt example

File: simple.txt

`<name_of_tag attribute='value'>Content<\name_of_tag>`



```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to my webpage</title>
</head>

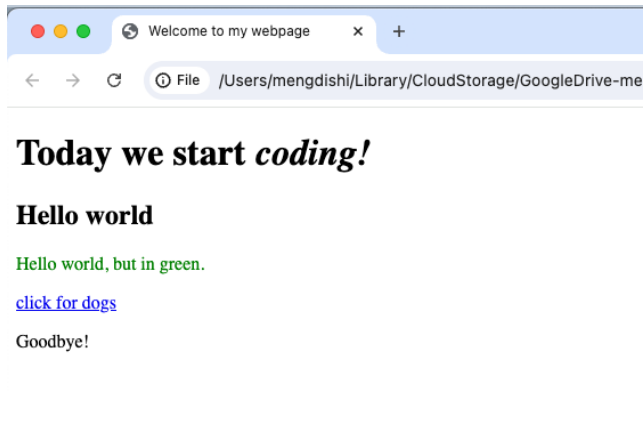
<body>
<h1 id='first'>Today we start <em>coding!</em></h1>
<h2 class='myclass'>Hello world</h2>
<p style="color: green;">Hello world, but in green.</p>

<p><a href='https://en.wikipedia.org/wiki/Dog'>click for
dogs</a></p>
<p id = 'second'> Goodbye!</p>

</body>
</html>
```

simple.txt example

If we open the simple.html file and right-click + “Inspect”...



simple.txt example

If we open the simple.html file and right-click + "Inspect"...

The screenshot shows a web browser window with the title "Welcome to my webpage". The address bar shows the file path: `/Users/mengdishi/Library/CloudStorage/GoogleDrive-mengdishi@uchicago.edu/My%20Drive/_Teaching/DAP%20II%20`. The page content is as follows:

Today we start *coding!*

Hello world

Hello world, but in green.

[click for dogs](#)

Goodbye!

The developer tools are open, showing the "Elements" tab. The DOM tree is expanded to the `<h2 class="myclass">Hello world</h2>` element, which is highlighted in blue. The corresponding HTML code is shown on the right:

```
<!DOCTYPE html>
<html>
  <head> ... </head>
  <body>
    <h1 id="first"> ... </h1>
    ... <h2 class="myclass">Hello world</h2> == $0
    <p style="color: green;">Hello world, but in green.</p>
    <p> ... </p>
    <p id="second"> Goodbye!</p>
  </body>
</html>
```

simple.txt example

Once we expand this, we get back simple.txt!



The screenshot shows a web browser window with the title "Welcome to my webpage". The address bar shows the file path: `/Users/mengdishi/Library/CloudStorage/GoogleDrive-mengdishi@uchicago.edu/My%20Drive/_Teaching/DAP%20II%20`. The webpage content is as follows:

Today we start *coding!*

Hello world

Hello world, but in green.

[click for dogs](#)

Goodbye!

The developer tools are open to the "Elements" tab, showing the following HTML structure:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Welcome to my webpage</title>
  </head>
  <body>
    <h1 id="first">
      "Today we start "
      <em>coding!</em>
    </h1>
    <h2 class="myclass">Hello world</h2>
    <p style="color: green;">Hello world, but in green.</p>
    ... <p> = $0
      <a href="https://en.wikipedia.org/wiki/Dog">click for dogs</a>
    </p>
    <p id="second"> Goodbye!</p>
  </body>
</html>
```

We can “inspect” any page on the web for its HTML

The screenshot shows a web browser displaying the Harris School of Public Policy website. The URL in the address bar is `harris.uchicago.edu/academics/design-your-path/specializations/specialization-data-analytics`. The page features a dark red header with navigation links: LOG IN, DIRECTORY, HIRE HARRIS, INFO FOR, MAKE A GIFT, and SEARCH. Below the header, the Harris School of Public Policy logo is visible, along with a secondary navigation bar containing links like About, Academics, Admissions, Student Life, Research & Impact, and News & Events. The main content area has a dark red background with a subtle geometric pattern. It includes a breadcrumb trail: HOME > ACADEMICS > DESIGN YOUR PATH > SPECIALIZATIONS. The title "Specialization in Data Analytics" is prominently displayed. Below the title, there is a sidebar with a list of specializations: Data Analytics Specialization (highlighted), Education Policy Specialization, Energy & Environmental Policy Specialization, and Finance & Policy. The main text area describes the specialization, noting its focus on data analysis in public policy and listing a requirement: "Write simple programs in Python".

Specialization in Data Analytics

Data Analytics Specialization

Education Policy Specialization

Energy & Environmental Policy Specialization

Finance & Policy

With increased digital access to data and the development of powerful, but inexpensive, computing, in the 21st century the formulation and evaluation of public policy is more and more reliant on the analysis of data. This specialization seeks to prepare students for careers where data analysis plays a central role. This specialization is designed for beginners without a prior background in coding and is a marker of courses passed rather than a competency determination.

Students who complete this specialization will be able to:

- Write simple programs in Python

Link: Harris Specialization in Data Analytics (also available as `harris_specialization.html`)

Note: the Inspect console will automatically focus on wherever you right click

We can “inspect” any page on the web for its HTML

The screenshot shows a web browser with the address bar displaying `harris.uchicago.edu/academics/design-your-path/specializations/specialization-data-analytics`. The page content on the left includes the University of Chicago Harris School of Public Policy logo and a section titled "Specialization in Data Analytics". Below this, a paragraph states: "With increased digital access to data and the development of powerful, but inexpensive, computing, in the 21st century the formulation and evaluation of public policy is more and more".

The right side of the image displays the browser's developer tools, specifically the "Elements" panel, which shows the HTML structure of the page. The code is partially expanded, revealing the following structure:

```
<!DOCTYPE html>
<html lang="en" dir="ltr" prefix="content: http://purl.org/rss/1.0/modules/content/ dc: http://purl.org/dc/terms/ foaf: http://xmlns.com/foaf/0.1/ og: http://ogp.me/ns# rdf: http://www.w3.org/2000/01/rdf-schema# schema: http://schema.org/ sioc: http://rdfs.org/sioc/ns# sioc: http://rdfs.org/sioc/types# skos: http://www.w3.org/2004/02/skos/core# xsd: http://www.w3.org/2001/XMLSchema# " class="js fa-events-icons-ready">
  <head>
    <body class="path-node page-node-type-general-page node--landing-page--secondary vsc-initialized">
      <!-- Google Tag Manager (noscript) -->
      <!-- End Google Tag Manager (noscript) -->
      <a href="#main-content" class="visually-hidden focusable skip-link"> Skip to main content </a>
      <noscript>
      <div class="dialog-off-canvas-main-canvas" data-off-canvas-main-canvas>
        <header id="global-navigation" class="header" xmlns="http://www.w3.org/1999/html">
          <div class="wrapper-content">
            ::before
            <div data-drupal-messages-fallback class="hidden"></div>
            <div id="block-currentnode" class="block block-harris-blocks block-harris-current-node-block">
              <div class="primary-page">
                <div class="container">
                  ::before
                  <main role="main" class="l-main">
                    <a id="main-content" tabindex="1"></a>
                    <div id="block-harris-theme-content" class="block block-system block-system-main-block">
                      <article class="node" data-history-node-id="14446" role="article" about="/academics/design-your-path/specializations/specialization-data-analytics">
                        <div class="node-content">
                          <section class="lb-section lb-section--25-75">
                            <div class="lb-section-row region-body">
                              <div class="lb-region lb-region--left">
                                <div class="lb-region lb-region--right">
                                  <div class="block block-layout-builder block-inline-blockregion-block-node-content">
                                    <div class="block block-layout-builder block-field-blocknodegeneral-pagebody">
                                      <div class="clearfix text-formatted field field--name-body field--type-text-with-summary field--label-hidden field-item">
                                        <p></p>
                                        <p>Students who complete this specialization will be able to:</p>

```

Link: Harris Specialization in Data Analytics (also available as `harris_specialization.html`)

Some Common HTML Tags

HTML **tags** always have the structure:

```
<open> ... </close>
```

Some Common HTML Tags

HTML **tags** always have the structure:

```
<open> ... </close>
```

- ▶ Headings: `<h1> ... </h1>`, `<h6> ... </h6>`
- ▶ Bold, italic: ` ... `, `<i> ... </i>`
- ▶ Paragraph: `<p> ... </p>`
- ▶ Hyperlinks `<a> ... `
- ▶ Images: ` ... `

Some Common HTML Attributes

HTML **attributes** always have the following structure:

```
<TAG attribute = 'attributevalue'> ... </TAG>
```

Some Common HTML Attributes

HTML **attributes** always have the following structure:

```
<TAG attribute = 'attributevalue'> ... </TAG>
```

- ▶ ID: <TAG id = 'idvalue'> ... </TAG>
- ▶ Class: <TAG class = 'classname'> ... </TAG>
 - ▶ Website elements often have unique IDs and classes, which are used to categorize types of content
 - ▶ We don't have to know when to use an id vs. class attribute – we just have to know how to scrape them

Some Common HTML Attributes

HTML **attributes** always have the following structure:

```
<TAG attribute = 'attributevalue'> ... </TAG>
```

- ▶ ID: `<TAG id = 'idvalue'> ... </TAG>`
- ▶ Class: `<TAG class = 'classname'> ... </TAG>`
 - ▶ Website elements often have unique IDs and classes, which are used to categorize types of content
 - ▶ We don't have to know when to use an id vs. class attribute – we just have to know how to scrape them
- ▶ Style: `<TAG style = 'color:red;'> ... </TAG>`

Some Common HTML Tags and Attributes

- ▶ Some tags and attributes are **commonly used together**
- ▶ Image + source:

```
<img src = 'image.png'>... </img>
```

img is the tag while src is the attribute (source path for the image file)

Some Common HTML Tags and Attributes

- ▶ Some tags and attributes are **commonly used together**
- ▶ Image + source:

```
<img src = 'image.png'>... </img>
```

img is the tag while src is the attribute (source path for the image file)

- ▶ Links:

```
<a href = 'www.google.com'> ... </a>
```

Some Common HTML Tags and Attributes

- ▶ Some tags and attributes are **commonly used together**
- ▶ Image + source:

```
<img src = 'image.png'>... </img>
```

img is the tag while src is the attribute (source path for the image file)

- ▶ Links:

```
<a href = 'www.google.com'> ... </a>
```

- ▶ Attributes can also be combined:

```
<img src = 'image.png' width = 500 height = 600> ... </img>
```

- ▶ This combines 1 tag (img) with 3 attributes (src, width, and height)

Do-pair-share

1. Inspect the HTML code on the Harris Specialization in Data Analytics ([link](#)) page
2. What is the tag associated with the text that starts with: “Students in the Master of Science in Computational Analysis and Public Policy...”?
3. What are examples of other content associated with the same tag?
4. Look at attributes for tags with links. Some of them do not appear to be “full” links. Can you explain why?

HTML to web scraping

- ▶ Webscraping steps:
 0. ~~Manual~~: inspect website's HTML to see how the info we want to extract is structured
 1. *Code*: download and save HTML code associated with a URL
 2. *Code*: parse through HTML code to extract information based on what we learned in Step 0 + refine
 3. *Code*: organize and save extracted information
- ▶ Now we can use code to do Steps 1-3!

Summary

- ▶ All websites are built in HTML – to web scrape, we need to know how to read it in order
- ▶ HTML has 3 elements: tags, attributes, and content
- ▶ “Inspect” a website to view its HTML

Using BeautifulSoup to Parse HTML

Roadmap

- ▶ Introduce BeautifulSoup library
- ▶ Example using `simple.txt`
- ▶ Demo how to extract URLs (used in webscraping later)

BeautifulSoup

- ▶ BeautifulSoup library: takes in HTML code and parses it in a structured way
- ▶ *Aside: the name Beautiful Soup is a reference to poorly-structured HTML code, which is called “tag soup”*
- ▶ In Terminal: `pip install bs4`, `pip install requests`, and `pip install lxml`

```
import pandas as pd
import requests
from bs4 import BeautifulSoup
```

BeautifulSoup

- ▶ `requests` allows you to open webpages. Usually use with URLs but in this case, we'll use it with a file on disk

```
with open(r'files/simple.html', 'r') as page:  
    text = page.read()
```

- ▶ The soup object is the website content, parsed into an easy-to-use reference
- ▶ `lxml` is an external resource used by browsers to parse HTML

```
soup = BeautifulSoup(text, 'lxml')
```

BeautifulSoup

```
print(text)
```

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<title>Welcome to my webpage</title>
```

```
</head>
```

```
<body>
```

```
<h1 id='first'>Today we start <em>coding!</em></h1>
```

```
<h2 class='myclass'>Hello world</h2>
```

```
<p style="color: green;">Hello world, but in green.</p>
```

```
<p><a href='https://en.wikipedia.org/wiki/Dog'>click for dogs</a></p>
```

```
<p id = 'second'> Goodbye!</p>
```


BeautifulSoup

```
print(soup)
```

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to my webpage</title>
</head>
<body>
<h1 id="first">Today we start <em>coding!</em></h1>
<h2 class="myclass">Hello world</h2>
<p style="color: green;">Hello world, but in green.</p>
<p><a href="https://en.wikipedia.org/wiki/Dog">click for dogs</a></p>
<p id="second"> Goodbye!</p>
</body>
</html>
```

Searching a soup object

- ▶ At first glance, soup object is very similar to the HTML code itself
- ▶ But it has “parsed” the code, making it **searchable** by tag and attribute

Searching a soup object

- ▶ `.find_all()`: searches for and returns list of *all* elements with a given tag

```
soup.find_all('p')
```

```
[<p style="color: green;">Hello world, but in green.</p>,  
<p><a href="https://en.wikipedia.org/wiki/Dog">click for dogs</a></p>,  
<p id="second"> Goodbye!</p>]
```

Searching a soup object

- ▶ `.find_all()`: searches for and returns list of *all* elements with a given tag

```
soup.find_all('p')
```

```
[<p style="color: green;">Hello world, but in green.</p>,  
<p><a href="https://en.wikipedia.org/wiki/Dog">click for dogs</a></p>,  
<p id="second"> Goodbye!</p>]
```

- ▶ Individual elements of the `.find_all()` output can be accessed via subscript

```
soup.find_all('p')[2]
```

```
<p id="second"> Goodbye!</p>
```

Searching a soup object

- ▶ `.find_all()`: searches for and returns list of *all* elements with a given tag

```
soup.find_all('p')
```

```
[<p style="color: green;">Hello world, but in green.</p>,  
<p><a href="https://en.wikipedia.org/wiki/Dog">click for dogs</a></p>,  
<p id="second"> Goodbye!</p>]
```

- ▶ Individual elements of the `.find_all()` output can be accessed via subscript

```
soup.find_all('p')[2]
```

```
<p id="second"> Goodbye!</p>
```

- ▶ A similar method: `.find()` searches for the *first* instance of an open/close tag

```
soup.find('p')
```

```
<p style="color: green;">Hello world, but in green.</p>
```

Searching a soup object

- ▶ We can use **lambda functions** with `.find_all()` to search for more specific things (refresher in mini-lesson this week)

```
soup.find_all('p', id = lambda h: h!=None and 'second' in h)
```

- ▶ Within all elements with tag p
- ▶ This lambda functions searches for elements:
 1. Where the id attribute is non-missing: `h != None`
 2. And the id tag includes the term 'second'

Searching a soup object

- ▶ We can use **lambda functions** with `.find_all()` to search for more specific things (refresher in mini-lesson this week)

```
soup.find_all('p', id = lambda h: h!=None and 'second' in h)
```

- ▶ Within all elements with tag p
- ▶ This lambda functions searches for elements:
 1. Where the id attribute is non-missing: `h != None`
 2. And the id tag includes the term 'second'

Results:

```
[<p id="second"> Goodbye!</p>]
```

- ▶ *Aside: writing this lambda code is a fantastic use case for ChatGPT! Don't get stuck on developing your lambda function*

Searching a soup object

- ▶ The output of `.find_all()` is always a *list* of objects, even if there's only one element in the list

```
soup.find_all('a')
```

```
[<a href="https://en.wikipedia.org/wiki/Dog">click for dogs</a>]
```

- ▶ The brackets that indicate that it's a list

Searching a soup object

- ▶ The output of `.find_all()` is always a *list* of objects, even if there's only one element in the list

```
soup.find_all('a')
```

```
[<a href="https://en.wikipedia.org/wiki/Dog">click for dogs</a>]
```

- ▶ The brackets that indicate that it's a list
- ▶ In contrast: `.find()` outputs just the object

```
soup.find('a')
```

```
<a href="https://en.wikipedia.org/wiki/Dog">click for dogs</a>
```

tag object and methods

- ▶ The output of `.find_all()` is a **tag** object

```
tag = soup.find_all('h1')[0]
print(type(tag))
```

```
<class 'bs4.element.Tag'>
```

tag object and methods

- ▶ The output of `.find_all()` is a **tag** object

```
tag = soup.find_all('h1')[0]
print(type(tag))
```

```
<class 'bs4.element.Tag'>
```

- ▶ The tag object is aware of its tag and attributes

```
tag.name
```

```
'h1'
```

```
tag.attrs
```

```
{'id': 'first'}
```

tag object and methods

- ▶ Most usefully for web scraping: it is aware of its contents.
- ▶ `tag.contents` returns a list of content (including nested tags)

```
tag.contents
```

```
['Today we start ', <em>coding!</em>]
```

```
print(type(tag.contents[0]))  
print(type(tag.contents[1]))
```

```
<class 'bs4.element.NavigableString'>
```

```
<class 'bs4.element.Tag'>
```

tag object and methods

- ▶ Most usefully for web scraping: it is aware of its contents.
- ▶ `tag.contents` returns a list of content (including nested tags)

```
tag.contents
```

```
['Today we start ', <em>coding!</em>]
```

```
print(type(tag.contents[0]))  
print(type(tag.contents[1]))
```

```
<class 'bs4.element.NavigableString'>
```

```
<class 'bs4.element.Tag'>
```

- ▶ While `tag.text` returns just the string inside

```
tag.text
```

```
'Today we start coding!'
```

tag objects can be nested

- ▶ We can apply BeautifulSoup methods to tag objects as well
- ▶ Example: this “p” tag object has an “a” tag nested within it

```
p_tag = soup.find_all('p')[1]
p_tag
```

```
<p><a href="https://en.wikipedia.org/wiki/Dog">click for dogs</a></p>
```

- ▶ If we wanted to get to the text inside the a tag, we have to apply `.find()` *again* to `p_tag`

```
a_tag = p_tag.find('a')
a_tag.text
```

```
'click for dogs'
```

tag objects methods can be used iteratively

► A more direct way of getting to the same text:

```
soup.find_all('p')[1].find('a').text
```

```
'click for dogs'
```

Extracting attribute value of tag object

- ▶ Say instead of the text, we were interested in the associated *URL*, which is contained in the attribute, href

```
print(a_tag.attrs)
```

```
{'href': 'https://en.wikipedia.org/wiki/Dog'}
```

- ▶ Output of `.attrs` is a *dictionary* with key href

Extracting attribute value of tag object

- ▶ Say instead of the text, we were interested in the associated *URL*, which is contained in the attribute, href

```
print(a_tag.attrs)
```

```
{'href': 'https://en.wikipedia.org/wiki/Dog'}
```

- ▶ Output of `.attrs` is a *dictionary* with key href
- ▶ So to access the URL, we use the `.get()` method

```
print(a_tag.get('href'))
```

```
https://en.wikipedia.org/wiki/Dog
```

- ▶ This forms the basis of *web crawling*, which we will next class

Summary

- ▶ Read in HTML code with `open()` from `requests`
- ▶ Then use `BeautifulSoup` parses HTML code into nested “tag” objects
- ▶ Key methods from `BeautifulSoup`:
 - ▶ `.find_all()`: return list of all tags
 - ▶ `.attrs`, `.contents`, `text` retrieve contents or attributes
- ▶ Key for web crawling: use `.get('href')` to get URLs associated with “a” tags

Example: Applying BeautifulSoup to a Website

Roadmap

- ▶ Applying BeautifulSoup methods to a real website
- ▶ The tricky part is specifying the combination of tags and attributes we want

Example with Harris Website

- ▶ Now let's try pulling an actual website's HTML code and navigating it with BeautifulSoup
- ▶ First, make a get request from Harris Specialization in Data Analytics page

```
url =  
    ↪ 'https://harris.uchicago.edu/academics/design-your-path/specialization  
response = requests.get(url)
```

Example with Harris Website

- ▶ Now let's try pulling an actual website's HTML code and navigating it with BeautifulSoup
- ▶ First, make a get request from Harris Specialization in Data Analytics page

```
url =  
    ↪ 'https://harris.uchicago.edu/academics/design-your-path/specializations/  
response = requests.get(url)
```

- ▶ Convert into a soup object

```
soup = BeautifulSoup(response.text, 'lxml')
soup.text[0:50]
```

[illegible]

- ▶ soup object is the text of the HTML code for this page

Example with Harris Website

Let's say we're interested in scraping all the bulleted items like the following:

- Write simple programs in Python
- Learn modern tools for data management, analysis, and presentation, including github, matplotlib, pandas, R, and SQL
- Construct and clean data sets from disparate sources and understand how to summarize and visualize modern data sets
- Use modern, computationally intensive methods to analyze data for the evaluation of policy
- The specialization's menu of electives is designed to allow students to increase their exposure to analytical methods used in the evaluation of public policy.

► *Right click + Inspect...*

Example with Harris Website

- Upon manual inspection of the HTML code, they appear to be under the tag

```
<p>Students who complete this specialization will be able to:</p>
<ul>
  <li>
    ::marker
    "Write simple programs in Python"
  </li>
  <li>
    ::marker
    "Learn modern tools for data management, analysis, and presentation, including github, matplotlib, pandas, R, and SQL"
  </li>
  <li> == $0
    ::marker
    "Construct and clean data sets from disparate sources and understand how to summarize and visualize modern data sets"
  </li>
  <li> ... </li>
  <li> ... </li>
```


Example with Harris Website

- ▶ Use `.find_all()` and then sanity-check the output

```
tag = soup.find_all('li')  
len(tag)
```

262

Example with Harris Website

- ▶ Use `.find_all()` and then sanity-check the output

```
tag = soup.find_all('li')  
len(tag)
```

262

- ▶ `.find_all()` has found 266 `li` tags in the HTML code
- ▶ That is much more than the total number of bullet points we're looking for!

Example with Harris Website

- ▶ To see what's going on, we can inspect the first few elements in the tag object

Example with Harris Website

- ▶ To see what's going on, we can inspect the first few elements in the tag object

```
tag[0:2]
```

```
[<li class="utility-navigation__item">  
  <a class="utility-navigation__item-link" data-drupal-link-system-path="us  
  </li>,  
  <li class="utility-navigation__item">  
    <a class="utility-navigation__item-link" data-drupal-link-system-path="di  
    </li>]
```

Example with Harris Website

- ▶ To see what's going on, we can inspect the first few elements in the tag object

```
tag[0:2]
```

```
[<li class="utility-navigation__item">  
  <a class="utility-navigation__item-link" data-drupal-link-system-path="us  
  </li>,  
  <li class="utility-navigation__item">  
    <a class="utility-navigation__item-link" data-drupal-link-system-path="di  
    </li>]
```

- ▶ To decide where to go next, have to think about what differentiates the elements we want vs. the elements we're getting
- ▶ The tag object is picking up elements that have another tag nested in them
- ▶ But from earlier, we know the bullet points we're interested in don't have anything nested in them

Example with Harris Website

- ▶ Recall that if a tag has something nested in it, we can apply `.find_all()` to it again
- ▶ So we can refine our search to exclude any elements that have another tag nested in them

```
li_nochildren = soup.find_all(lambda t: t.name == 'li' and not  
    ↪ t.find_all())
```

- ▶ lambda function looks for `li` tags that have **nothing nested in them**
- ▶ *Aside: since `t.contents` returns a list, we can't use `t.contents == None`*

Example with Harris Website

- ▶ Recall that if a tag has something nested in it, we can apply `.find_all()` to it again
- ▶ So we can refine our search to exclude any elements that have another tag nested in them

```
li_nochildren = soup.find_all(lambda t: t.name == 'li' and not  
    ↪ t.find_all())
```

- ▶ lambda function looks for `li` tags that have **nothing nested in them**
- ▶ *Aside: since `t.contents` returns a list, we can't use `t.contents == None`*
- ▶ Sanity-check the length: this seems to have eliminated many of the `li` tags that we didn't want

```
len(li_nochildren)
```

Example with Harris Website

- ▶ We can then extract the content from this tag object into a list using `.contents`

```
li_nochildren_content = [li.contents for li in li_nochildren]
```


Example with Harris Website

- ▶ Inspecting the beginning of the list ...

```
for item in li_nochildren_content[0:4]:  
    print(item)
```

```
['\n                Specialization in Data Analytics\n                ']  
['Write simple programs in Python']  
['Learn modern tools for data management, analysis, and presentation, including  
['Construct and clean data sets from disparate sources and understand how to
```

Example with Harris Website

- ▶ Inspecting the beginning of the list ...

```
for item in li_nochildren_content[0:4]:  
    print(item)
```

```
['\n                Specialization in Data Analytics\n                ']  
['Write simple programs in Python']  
['Learn modern tools for data management, analysis, and presentation, including  
['Construct and clean data sets from disparate sources and understand how to
```

- ▶ and the end...

```
for item in li_nochildren_content[-4:-1]:  
    print(item)
```

```
['PPHA 42000 Applied Econometrics I\xa0']  
['PPHA 42100 Applied Econometrics II\xa0']  
['PPHA 60000 Policy Labs\xa0(with permission of the Specialization Director
```

Example with Harris Website

- Do some final cleanup to remove the first element, which is not a bullet point

```
li_nochildren_content = li_nochildren_content[1:]  
for item in li_nochildren_content[0:5]:  
    print(item)
```

```
['Write simple programs in Python']  
['Learn modern tools for data management, analysis, and presentation, including  
['Construct and clean data sets from disparate sources and understand how to  
['Use modern, computationally intensive methods to analyze data for the evaluation  
["The specialization's menu of electives is designed to allow students to : ]
```

Example with Harris Website

Final webscraping code:

```
# 1. Extracts and saves HTML code as a parseable object
url =
    ↪ 'https://harris.uchicago.edu/academics/design-your-path/specializations/specializations'
response = requests.get(url)
soup = BeautifulSoup(response.text, 'lxml')

# 2. Specifies tags and attributes we want to collect
li_nochildren = soup.find_all(lambda t: t.name == 'li' and not
    ↪ t.find_all())

# 3. "Scrapes" elements from HTML code based on step 2
li_nochildren_content = [li.contents for li in li_nochildren][1:]

# 4. Final cleanup
li_nochildren_content = li_nochildren_content[1:]
```

Summary: generalizing from this example

- ▶ Step 1 of a scraper (requesting and extracting HTML) will almost always be the same
- ▶ The “hard” part of writing a web scraper is step 2: identifying the tags and attributes we want to collect
 - ▶ Steps 2 and 3 have to be uniquely tailored to each specific task and website
 - ▶ Ironing out 2 and 3 may require several iterations of going back and forth between your code and manually parsing